Manipulate web content for embedded devices

Pulling the Rug

If you control the Ethernet line or the wireless router, you can inject new content into connected devices or make them believe they are on the other side of the world, even if they are reluctant to cooperate. *By Mike Schilli*

I have had a Chumby [2] (Figure 1) on my desk for about a year now; you may be familiar with these radio/alarm clock-shaped boxes that show you a ticker with the latest headlines, weather, share prices, and news from the world of programming, thanks to a collection of configurable apps.

Strangely, this secondary miniscreen, which I tend to follow out of the corner of my eye, is less distracting than a widget on my main screen, and it lets me stay abreast of the latest Silicon Valley gossip while I'm working or even witness the birth of seminal developments from the gadget industry.

Going DEFCON-2 on Ads

A few weeks ago, somebody at Chumby HQ led astray by the temptations of advertising revenue decided to squeeze a 10-second spot for car insurance in between the channels I subscribe to (Figure 2).

I immediately went to DEFCON-2 [3] to find out if the Chumby's settings support an HTTP proxy that would give me an approach to catching and knocking out unsolicited ads before they reached my Chumby.

A visit to the Chumby forum showed that Chumby does not support HTTP proxies by default, but you can plug in a memory stick (Figure 3) with the proxy settings (Figure 4) stored in the root directory in a file named `userhook0`. When you boot with the memory stick plugged in, the device will then forward HTTP requests through the proxy server, which can check and, if needed, manipulate the URLs.

Squid or Home-Grown?

You can use Squid [4] as the proxy server, but if you enjoy a spot of tinkering, you can put together a logging proxy based on the CPAN HTTP::Proxy module, as shown in Listing 1. Line 29 launches the proxy server, which listens

MIKE SCHILLI

Mike Schilli works as a software engineer with Yahoo! in Sunnyvale, California. He can be contacted at *mschilli@perl-meister.com*. Mike's homepage can be found at *http://perlmeister.com*.

Figure 1: The Chumby displaying the latest headlines from ihackernews.com.

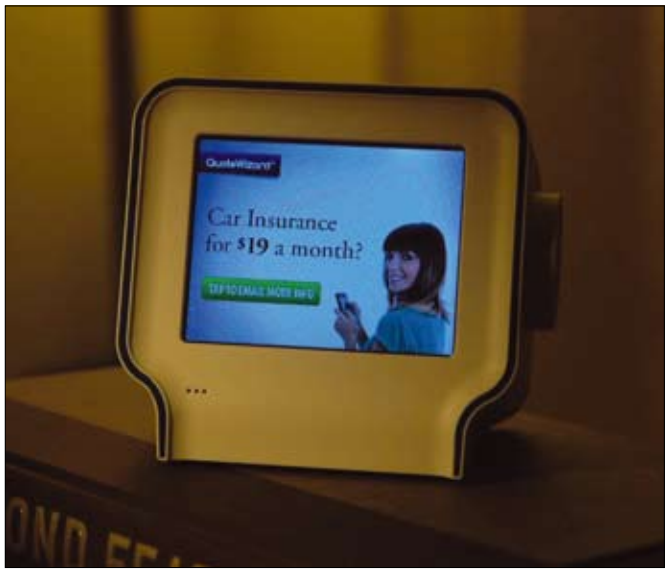

Figure 2: The wicked witch of the ad west gate-crashing my Chumby channels.

Figure 3: The USB hub connects my Chumby with an Ethernet adapter and holds the memory stick with the user hook file that contains the proxy settings (Figure 4).

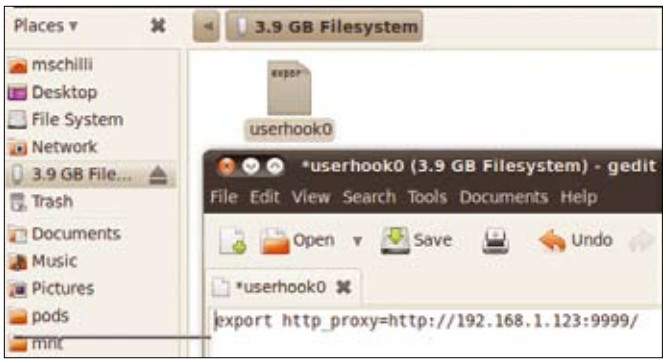

Figure 4: If you plug a memory stick with a `userhook0` file into your Chumby, the device will load the proxy parameters set in the file at boot time.

on port 9999 of the host at 192.168.1.123, accepts the Chumby's HTTP requests, and retrieves the content from the Internet.

The `request` filter in line 17 looks at the incoming HTTP request, extracts the URL, and outputs the URL with `print`. Figure 5 shows you what the Chumby is doing, thanks to the man in the middle. To pull the carpet from under the Chumby adman's feet, Listing 2 changes the `push_filter` to rewrite the URL of the request if the Chumby asks for `eviladserver.com`.

Now, instead of seeing annoying ads, I get to see an image of Sandy Beach on my favorite Hawaiian island, Oahu, with bodysurfers enjoying the breakers.

```
mschilli.mybox/eg-master$ ./proxy-logger
http://ping.chumby.com/setecastronomy
http://www.chumby.com/crossdomain.xml
http://www.chumby.com/crossdomain.xml
http://www.chumby.com/xml/controlpanel?id=68449453-0E38-53D8-0235-9153C8E2AA9D&hw=10.8&sw=1.0.7&fw=1.0.3454&dcid_vers=0002&dcid_rgin=0001&dcid_skin=0001&dcid_part=1000&dcid_camp=0002&lang=en-us
http://s3.chumby.com/control_panels/controlpanel2.8.79.swf
http://www.chumby.com/crossdomain.xml
http://xml.chumby.com:80/manifest/show?id=68449453-0E38-53D8-0235-9153C8E2AA9D&hw=10.8&sw=1.0.7&fw=1.0.3454&id=68449453-0E38-53D8-0235-91AFC8E2AA9D&dcid_camp=0002&dcid_part=1000&dcid_skin=0001&dcid_rgin=0001&dcid_vers=0002&config=falconwing&timezone=-0800
http://xml.chumby.com:80/xml/authorize?hw=10.8&sw=1.0.7&fw=1.0.3454&id=68449453-0E38-53D8-0235-9153C8E2AA9D&nocache=1323412536807&config=falconwing
http://content.chumby.com:80/music_sources/show/?hw=10.8&sw=1.0.7&fw=1.0.3454&id=68449453-0E38-53D8-0235-9153C8E2AA9D&nocache=10928&dcid_camp=0002&dcid_part=1000&dcid_skin=0001&dcid_rgin=0001&dcid_vers=0002&config=falconwing
```

Figure 5: The logging proxy keeping a record of the URLs that the Chumby calls.

Manipulating Proxy

In line 25, the HTTP::Proxy framework passes in three parameters to the filter callback: `$self` contains a reference to the filter object itself; `$headers` contains an HTTP header object, and `$req` has the HTTP request object. If the pattern match in line 28 determines that the request URL points to `eviladserver.com`, line 31 sets the request object of the calling function in `$_[2]` to a new URL, which points to the JPG image of the Hawaiian beach on *perlmeister.com*. A brief test with Firefox and the proxy settings in Figure 6 confirm the redirect, as shown in Figure 7.

Of course, some devices won't let the end user set a proxy. Take the Chumby, for example; some apps use a library that simply ignores the proxy settings. If

Mike: Figure 9 callout OK in "When the Kindle" paragraph??

Figure 6: The Firefox Connection Settings dialog points the browser at the modifying proxy.

Figure 7: Firefox with the proxy settings picking up the predefined image instead of the ads.

LISTING 1: proxy-logger

```
01 #!/usr/local/bin/perl -w
02 ############################
03 # proxy-logger
04 # Mike Schilli, 2012
05 #   (m@perlmeister.com)
06 ############################
07 use strict;
08 use HTTP::Proxy;
09 use
   HTTP::Proxy::HeaderFilter::simple;
10
11 my $proxy = HTTP::Proxy->new(
12  host => "192.168.1.123",
13  port => 9999
14 );
15
16 $proxy->push_filter(
17  request =>
18    HTTP::Proxy::HeaderFilter::
      simple->new(
19      sub {
20       my ($self) = @_;
21
22       print $self->proxy
23           ->request->uri(),
24           "\n";
25      }
26    )
27 );
28
29 $proxy->start;
```

you can't access the source code for the programs running on the device, you will need to resort to tricks in the various network layers to redirect the outgoing IP packets as you prefer.

The Taming of the Shrew

Another example is the Amazon Kindle, which will connect to a wireless network without asking you for a proxy server (Figure 8). But, what if the local wireless network needs a proxy or a tunnel to access the Internet or if you want to hide your true IP from the Amazon server?

When the Kindle connects with a wireless network, it picks up a dynamic IP from the DHCP server and is given the default gateway IP at the same time, typically the address of your home network router. When the Kindle issues a web request, it sends a packet addressed to the web server via the default gateway. If the gateway happens to be a Linksys router with Tomato firmware [5] in place, it can capture and manipulate the packets using the `iptables` instructions shown in Figure 9; luckily, the device runs Linux with a full-fledged network stack. The second rule checks whether an incoming packet is addressed to port

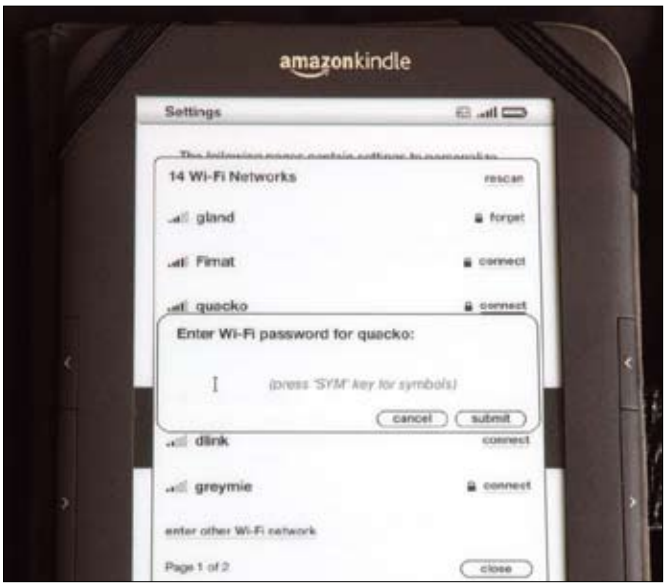

Figure 8: You can't set a proxy when the Kindle connects with the wireless network.

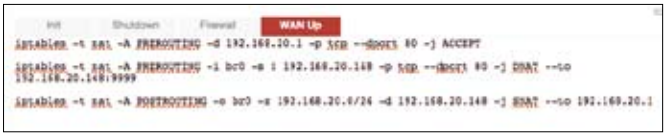

Figure 9: These `iptables` rules tell the Linksys router to redirect HTTP requests to the proxy.

LISTING 2: proxy-adkiller

```
01 #!/usr/local/bin/perl -w
02 ############################
03 # proxy-adkiller
04 # Mike Schilli, 2012
05 #   (m@perlmeister.com)
06 ############################
07 use strict;
08 use HTTP::Proxy;
09 use HTTP::Proxy::HeaderFilter::
            simple;
10 use HTTP::Request::Common;
11
12 my $proxy = HTTP::Proxy->new(
13  host => "192.168.1.123",
14  port => 9999
15 );
16
17 my $repl =
18   "http://perlmeister.com/test/"
19     . "sandy-beach.jpg";
20
21 $proxy->push_filter(
22  request =>
23  HTTP::Proxy::HeaderFilter::simple
    ->new(
24    sub {
25      my ($self, $headers,
26          $req)          = @_;
27
28       if ($req->uri() =~
29          /eviladserver\.com/)
30       {
31        $_[2] = GET $repl;
32       }
33      }
34    )
35 );
36
37 $proxy->start;
```

FULL
PAGE
AD

Mike: Should the temporary file name be /etc/ibvpn.conf as in Figure 11?? -rls

80 on any web server and, if so, redirects the packet to the address of the home-baked proxy server listening on port 9999 on a nearby Linux box.

The first instruction in Figure 9 avoids redirecting packets destined for the router's own web admin interface. It picks up the incoming IP packets addressed to port 80 on the router; the `-d` (for destination) points to the router IP of 192.168.20.1. In the `PREROUTING` phase, the rule sets the `ACCEPT` target so that the packets are sent directly to the router's web server without applying the subsequent `iptables` rules.

Tomato with sshd

You can also launch an `sshd` server in the Tomato firmware and log in to the device by typing `ssh root@192.168.20.1`. Once there, type the admin password that you use for the web interface, followed by `ifconfig` in the opening shell; this command reveals that the router's LAN interface answers to the name of `br0` and the WAN interface is named `eth0`.

The computer with the logging proxy server connected to the local network has an IP address of `192.168.20.148`; the second `iptables` instruction in Figure 9 thus redirects all packets that do not come from the proxy (`-s ! IP`) and are addressed to port 80 of any server to the proxy's IP and port 9999. This technique is known as transparent or intercepting proxying [6] because no settings are required on the client; in fact, the client is blissfully unaware that any proxying is going on at all.

Figure 10: The Tomato USB firmware [7] on the Linksys router defines an OpenVPN client that gives connected wireless clients a WAN address in a different geographical location.

To direct any return packets from the target web server via the router, rather than sending them straight to the original sender, the third `iptables` rule in Figure 9 rewrites the packets destined for the proxy so that they specify the router's address as their return address, rather than the address of the original web client.

The Tomato UI needs to tell the router to store the three rules persistently in its NVRAM memory and run them after a reboot once the router's WAN interface becomes available, so you need to store them in the *Administration* | *Scripts* tab *WAN Up*.

Now, when I connect a Kindle 3 with the wireless network provided by the router and type a couple of URLs in the browser located in *Experimental* | *Browser*, I can see the requests the Kindle browser issues in the proxy log.

Secure with SSL

If a sender uses an SSL connection with an `https://` *URL* and correctly implements the accompanying certificate checks, an intermediate proxy can't listen to or manipulate the communication. The SSL connection effectively prevents this man-in-the-middle attack.

Fortunately, you can redirect the IP packets via VPN and manipulate the NAT settings so the server contacted by the device thinks that the device is located at the VPN server's location. This is a useful approach for websites that refuse to deliver content to specific geolocations. The extended Tomato router firmware, `tomato-usb` [7], uses an OpenVPN client for this; it can quite easily set up a connection with a VPN provider like *ibvpn.com*, which offers OpenVPN servers in various countries for consideration (Figure 10).

The startup script in Figure 11 writes the username and the password for *ibvpn.com* to a temporary file by the name of `/tmp/ibvpn.conf` when the router boots. Because files on the router's filesystem are lost when you reboot, you can use a shell script in Tomato space to create the script stored in NVRAM dynamically when the router boots. The first command pushes the script content into a temporary file by the name of `/tmp/vpn.sh`, as Figure 11 shows, and the subsequent `sh /tmp/vpn.sh &` executes the shell code.

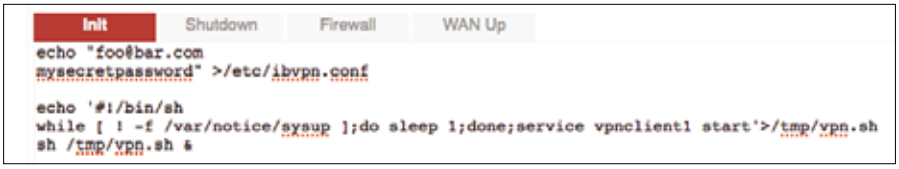

Figure 11: The scripts below *Administration* start the OpenVPN client when the router boots.

The script itself waits with a `while` loop and `sleep` until `/var/notice/sysup` exists. This is the router telling us that the system has booted successfully. Once these conditions are fulfilled, `service vpnclient1 start` launches the first OpenVPN client (Tomato USB gives you two), which then connects to the OpenVPN server.

For a successful handshake, you need the server's OpenSSL certificate in the *Keys* tab (Figure 10) on the Tomato USB as well as any secret keys that you use. If this works, the OpenVPN server will use a `push` command to configure the router's routing table so that outgoing packets are automatically sent to the other end of the VPN tunnel. From there, the OpenVPN server sends them to the target web server, which thinks it is talking to the OpenVPN server.

This method works perfectly for SSL connections, too. The little detour via the VPN tunnel goes unnoticed because packets may float freely and take any route they want without affecting the integrity of the connection at the application level. ■■■

INFO

[1] Listings for this article: *http://www.linux-magazine.com/Resources/Article-Code*

[2] Chumby: *http://chumby.com*

[3] Defense readiness condition: *http://en.wikipedia.org/wiki/DEFCON*

[4] Squid HTTP proxy: *http://www.squid-cache.org*

[5] Tomato firmware for the Linksys router: *http://www.polarcloud.com/tomato*

[6] Saini, Kulbir. *Squid Proxy Server 3.1: Beginner's Guide*. Packt Publishing, 2011

[7] Tomato USB firmware for the Linksys router with VPN software: *http://tomatousb.org*